# Structure Search and Stability Enhancement of Bayesian Networks

**Hanchuan Peng** and **Chris Ding**
NERSC Division, Lawrence Berkeley National Laboratory,
University of California, Berkeley, CA, 94720, USA
Email: hpeng@lbl.gov, chqding@lbl.gov

## Abstract

Learning Bayesian network structure from large-scale data sets, without any expert-specified ordering of variables, remains a difficult problem. We propose systematic improvements to automatically learn Bayesian network structure from data. First, we propose a linear parent search method to generate candidate graph. Second, we propose a formal approach to eliminate cycles using minimal likelihood loss, a short cycle first heuristic, and a cut-edge repairing. Third, we propose structure perturbation to assure the stability of the network. This step also suggests a stability-enhancement method to refine the network structure. The algorithms are easy to implement and efficient for large networks. Experimental results on two data sets show that our new approach outperforms existing methods.

## 1. Introduction

The rapidly increasing quantity of data in many data mining fields allows a great opportunity to model and understand the complicated relationship among a large number of variables. Bayesian Networks (BNs) [18,5,1, 11] provide a natural framework for this purpose [2,8,10,17,19,23,22,20]. For example, Murphy and Mian [17] used the dynamic BNs to infer genetic networks from time series data. Hartemink et al [10] presented an annotated edge method based on BNs to model regulatory networks. Sarkar and Boyer proposed Bayesian perceptual inference networks [22]. Peng et al used the BNs in morphology-function analysis of medical images [20]. Cheng applied BN [2] to biomedical data feature extraction and classification which won a recent data mining contest (KDD Cup-2001).

A Bayesian network [1,18,5,11] is a Directed Acyclic Graph (DAG) $G = (V, E)$ that models the probabilistic dependencies of a group of nodes, i.e.

$$P(\{g\}) = \prod_{g \in V} P(g \mid \pi_g), \qquad (1)$$

where each node $g$ stands for a variable, $\pi_g$ is the parents of node $g$ in $G$; the directed edges (or hyper-edges) among nodes encode the respective conditional probabilistic distributions. The factorization property allows the joint distribution of the network be factorized as the product of conditional probabilities of every variable given its parents.

A major difficulty is to identify BN structures from input data $D$ directly. The problem is NP-hard [3], since the total number of possible graphs is exponential to $n$, the number of variables in the data [5,21]. Many heuristic search methods (for reviews see Heckerman [11] and Buntine [1]) have been proposed. A well-known algorithm is K2 [5], which however assumes a predefined ordering of all nodes, thus is not suitable wherever there is little knowledge about variables' ordering. Many previous methods, e.g. Cheng's conditional independence test based method [2], often have at least $O(n^4)$ complexity. Another drawback of many existing methods is that predefined thresholds are necessary to terminate the BN search procedure (e.g. [19]). For many large applications, the number of variables is large, several hundreds at least. Thus an efficient algorithm to identify BN structures is particularly important.

Because of the difficulty in constructing BNs, an intuitive approach is to first draft candidate graphs and then eliminate possible cycles (e.g. [15,16,2]). Formally, a candidate graph is very similar to the dependency graph [12]. Our work emphasizes more on the efficiency in constructing BNs and the quality measures of the learnt BNs.

In this paper we propose a new $O(n^2)$ algorithm to infer locally stable Bayesian networks without requiring any predefined ordering of variables or predefined thresholds to terminate the model search. Our new algorithm consists of three main steps. First, we develop an efficient algorithm to search for optimal parents of nodes to form a candidate graph, which includes important gene regulations for every variable. We propose two improvements on the K2 algorithm to generate the candidate graph by searching for a minimum set of parents for every node with a linear search heuristic to maximize the posterior. This method significantly reduces the complexity of the Bayesian learning. (See §3)

Second, we propose a formal treatment for eliminating possible cycles in the candidate graph that would violate the acyclic assumption of BNs. We propose a minimal likelihood loss criterion and a short cycle first heuristic that can be efficiently implemented. We also

design a heuristic to repair the DAG structure due to cycle elimination. (See §4)

Third, we propose a locally stable condition as a desirable feature of BN: any local change of edges will lower the posterior of the network. We design structural perturbation scheme to evaluate networks. We also propose algorithm to enhance the stability. (See §5)

## 2. Bayesian Likelihood Score

With the assumption of a uniform prior of $G$ (for discussions on priors, see [11]), we use the logarithm posterior (and likelihood) of $G$ given the data $D$,

$$\ell(G) = \log P(G|D) \propto \log P(D|G) \qquad (2)$$

as the metric to judge the optimality of a network structure. The likelihood $P(D|G)$ can be calculated in the following [5]:

$$P(D|G) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i-1)!}{(N_{ij}+r_i-1)!} \prod_{k=1}^{r_i} N_{ijk}\,!\,, \qquad (3)$$

where $n$ is the number of variables, or nodes; $r_i$ is the number of states of the $i$th variable; $\pi_i$ is the set of parent nodes of the $i$th variable; $q_i$ is the the numbers of joint states of the nodes in $\pi_i$; $N_{ijk}$ is the number of cases in which the $i$th variable assumes the $k$th state and the set $\pi_i$ assumes the $j$th joint state; and $N_{ij} = \Sigma_{k=1}^{r_i} N_{ijk}$. Other scores include Minimal Description Length (MDL) [1], BDe [11], etc.

## 3. Candidate Graph

The candidate graph $G_c$ is a directed graph containing important associations of variables where the redundancy of associations should be minimized. Our approach is to identify the optimal parent set for each node based on the Bayesian likelihood. Here our emphasis is on how to efficiently search for optimal parent set, $\pi = \{g_i^*, i=1,\ldots,m\}$. The locally optimal parent set is similar to dependency graph of Heckerman *et al* [12]; the difference is that they used regression to determine the dependency while we directly search based on the Bayesian score.

Our algorithm improves upon the K2 search algorithm [5]. K2 algorithm uses a simple incremental search strategy: it first searches for the best singleton parent $g_1^*$, i.e., $g_1^* = \text{argmax}_i\ \ell(g_i \to g)$, and $\ell(g_i \to g) > \ell(g) + \ell(g_i)$. It then searches for the second parent according to (a) $g_2^* = \text{argmax}_i\ \ell(g_1^* \to g \leftarrow g_i)$ and (b) the new score is increased: $\ell(g_1^* \to g \leftarrow g_i) > \ell(g_1^* \to g) + \ell(g_i)$. If the second parent cannot be found, then the search is terminated; otherwise, continue to search for the third parent, etc. K2 has a natural termination and does not need a threshold.

We extend K2 in two directions. (i) We note that once a parent or parents are found, many of the rest of nodes are rendered conditionally independent of $g$. Thus in searching for second parent $g_2^*$, we do not need to search through all the rest variables, $\Omega_1 = \{V \setminus g_1^*\}$; instead we need only search

$$\Omega_1^+ = \{g_i \in V, g_i \neq g_1^*, \ell(g_i \to g) > \ell(g_i) + \ell(g)\}. \qquad (4)$$

Note that $\Omega_1^+$ is obtained automatically when searching for $g_1^*$. Similarly, when searching for $g_3^*$, we need only search $\Omega_2^+$, instead of $\Omega_2 = \{V \setminus \{g_1^*, g_2^*\}\}$; etc. This amounts to a large saving in computation.

(ii) We systematically search a larger space than K2. In K2, $g_1^*$ corresponds to the largest $\ell(g_i \to g)$. Denote the respective parent set as $\pi^{(1)}$. We can search another set of parents beginning with the second largest $\ell(g_i \to g)$, denoted as $\pi^{(2)}$. If $\pi^{(2)}$ leads to better score than $\pi^{(1)}$, then we take $\pi^{(2)}$ as the final parent set. We call this 2-max search. This can be extended to $k$-max search. Clearly, the Bayesian score of $\pi^{(k)}$ increases monotonically with $k$, at the expense of linearly enlarged complexity.

We call this modified method the K2+ algorithm. It has the complexity $O(\alpha kmn)$, where $\alpha$ counts for the reduction using $\Omega_1^+, \Omega_2^+, \ldots, \Omega_m^+$ instead of $\Omega_1, \Omega_2, \ldots, \Omega_m$ (often $\Omega_i^+$ contains a much smaller number of variables than $\Omega_i$). Accordingly, the complexity to construct the whole candidate graph is $O(n^2)$, which is much less than that of many other popular methods, e.g. Cheng's $O(n^4)$ algorithm [2].

## 4. Cycle Elimination

Since the candidate graph $G_c$ is generated via local optimal search, it is possible that $G_c$ contains many cycles that violate the basic acyclic assumption of BNs.

A simple approach is to enumerate all possible DAGs that could emerge from $G_c$ and select the one with the largest score. However, this method is impractical due to its exponential complexity. One earlier approach is to use a predefined ordering of all nodes to partially avoid this problem, as used in many BN learning algorithms [5,1,7]. Approximation methods based on random edge cut [16] have also been studied. A heuristic decision-tree based approach has also been studied in [15]. It seems, however, that a formal study on cycle elimination is lacking so far. In this paper, we resolve this problem via graph algorithmic approach. This approach consists of three heuristics that can be implemented efficiently.

Any cycle must lie in a Strongly Connected Component (SCC) of the graph. An efficient $O(n)$ algorithm based on depth-first search can locate SCCs in a di-

rected graph [6]. We first find all the SCCs in $G_c$, and eliminate cycles only within each SCC.

### 4.1 Bayesian Likelihood Loss Function

If a SCC contains one cycle, we can break one cycle at a time. We break cycles based on loss function. For each edge $g_i \to g_j$, we define the loss as the reduction of Bayesian log-likelihood for $g_j$ due to the loss of one of its parents,

$$w(g_i \to g_j) = \ell(g_j \mid \pi) - \ell(g_j \mid \pi \setminus g_i). \quad (5)$$

Note that $w(g_i \to g_j) \neq w(g_i \leftarrow g_j)$. Although mutual information might be another possible choice as the loss, it does not reflect the joint association between different parents and $g_j$.

If a SCC contains several cycles, sometimes they share one or more common edges, such as the cycles in Figure 1. For example, in Figure 1 (a) the edge $g_2 \to g_3$ is shared by the cycles $C_{1231}$ (i.e. $g_1 \to g_2 \to g_3 \to g_1$) and $C_{2342}$.

There are several criteria to break the cycles. (i) We can simply cut edges with the smallest loss. (ii) We can identify the common edges and cut the one shared by most cycles. In Figure 1 (b), cutting the common edge $g_2 \to g_3$ will eliminate two cycles. (iii) The loss function criterion indicates there could be better choices. Suppose $g_1 \to g_2$ and $g_3 \to g_4$ are the edges with the minimal loss in cycles $C_{1231}$ and $C_{2342}$, respectively. If the condition

$$w(g_1 \to g_2) + w(g_3 \to g_4) < w(g_2 \to g_3) \quad (6)$$

holds, then we break edges $g_1 \to g_2$ and $g_3 \to g_4$; otherwise, we break the edge $g_2 \to g_3$.

Figure 1 (b) illustrates a more complicated SCC with four 3-node cycles $C_{2312}$, $C_{2342}$, $C_{2542}$, $C_{2642}$. The edge $g_2 \to g_3$ is shared 2 times and the edge $g_4 \to g_2$ is shared 3 times. We start cycle elimination from the most-shared edge (i.e. $g_4 \to g_2$) and use a minimal-likelihood-loss strategy similar to Eq.(6). If the edge $g_4 \to g_2$ is cut, then only $C_{2312}$ remains and we will further cut its minimal loss edge; otherwise we use Eq.(6) to decide which edge(s) in cycles $C_{2312}$ and $C_{2342}$ should be broken.

This minimal-likelihood-loss criterion can be summarized as follows. If there is no nested cycle, for each cycle we break the edge with the minimal loss. When several cycles nest among themselves, we identify the edge $e_{ij}$ shared by most cycles and compare its loss with the sum of the minimal loss edges in participating cycles; if breaking $e_{ij}$ leads to less loss, we will cut $e_{ij}$; otherwise we cut the minimal loss edges in every participating cycle.

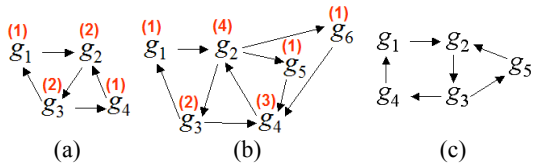


*Figure 1.* (a) A SCC with two 3-node cycles. (b) A SCC with four 3-node cycles. (c) A SCC with a 3-node cycle and a 4-node cycle. Multiplicities of nodes are shown in parentheses.

### 4.2 Short-Cycle-First Heuristic

Although the minimal likelihood loss criterion is generally applicable, finding the set of edges with regarding to this criterion could become very complicated due to the existence of many cycles and the large number of common edges they share. Here we propose a short-cycle-first heuristic to minimize the complexity (for both computation and implementation) of cycle elimination.

In BNs, information propagates multiplicatively because of the probability calculation, thus along a fixed path of $m$ edges, the influence of the starting node on the ending node is $P_1 P_2 \cdots P_m$ approximately. Therefore, usually a long cycle violates the acyclic assumption less severely than a short cycle. If a SCC contains cycles of different lengths, our short-cycle first heuristic breaks the 2-node cycle first, and the 3-node cycle second, etc. In Figure 1 (c), we first break the 3-node cycle $C_{2352}$. Afterwards, we break the cycle $C_{12341}$ if it still exists.

Short-cycle-first heuristic can be efficiently implemented through a matrix multiplication method. Let $A$ be the adjacency matrix of a SCC. Diagonal elements of $A$ are zeros. We compute $A^m$ with the smallest $m$ such that nonzero elements appear at matrix diagonal; with some elementary algebra [9], we can show that (i) nodes corresponding to nonzero diagonal elements in $A^m$ must involve in $m$-node cycles; thus finding these cycles are restricted to the subgraph induced by these nodes; (ii) the multiplicity of node $i$ (i.e. value of $(A^m)_{ii}$) equals the number of times a cycle pass through node $i$ (for example, in Figure 1 (a) and (b) the multiplicity of nodes are indicated by red numbers in parentheses). (iii) Starting from the node with the highest multiplicity using breadth-first-search algorithm, restricting on the subgraph, we can easily traverse all $m$-node cycles and identify the most-shared edges. For example, in Figure 1 (b), we can start from $g_2$ and quickly identify the most-shared edge $e_{42}$. We use the criterion in §4.1 to break cycles. Note that $A$ is usually very sparse and the sparse matrix multiplications often involve much less computation than dense matrixes.

The short-cycle-first heuristic allows us to solve much simpler cycle elimination problems, with the

minimal loss criterion. Once one or more edges are cut, we re-run the SCC-detection algorithm to identify the new (probably smaller) SCCs and the matrix multiplication method to identify all remaining cycles. This is repeated until all cycles are eliminated. Eventually this procedure returns a DAG $G$.

### 4.3 Repair of Local Structures

Once an edge $g_i \rightarrow g_j$ in the candidate graph $G_c$ is cut, there is a loss of the likelihood of $\ell(g_j|\pi_j)$ because now $g_j$'s parent set $\pi_j$ is less optimal. Hence, we must repair the parent set of each node whose incident edges have been cut. K2+ parent-search is used to find the optimal parent set. The repair is done locally, i.e., all other parents of $g_j$ are retained during the repair of $\pi_j$. In addition, the repair is subject to the acyclic condition, i.e. the best replacement edge cannot cause cycles.

Suppose in cycle elimination, $M$ edges are cut and the local structures of the involved nodes need repair. The sequential order in repairing local structures is important, because the first-repaired local structures will give extra-constraints on the space of the later-repaired local structures due to the acyclic condition (i.e. potentially the search-space of the later-repaired local structures would be shrunk). Our algorithm proceeds as the following.

By comparing the candidate graph $G_c$ and the DAG $G$ returned from cycle elimination, we first locate the nodes whose local structures need repair. We calculate the likelihood loss of a node due to the cutting of incident edges. We sort these loss values from large to small, and repair the nodes according to this ordering. This maximal-loss-first heuristic is consistent with the minimal-likelihood-loss criterion. Clearly, during the course of repair, the DAG after each local repair will always have a higher likelihood score than the DAG before this local repair. This repair algorithm has the complexity of $O(\beta n)$, where $\beta$ is the number of nodes whose parent-sets are repaired.

For simplicity, we call the whole paradigm (including the minimal likelihood loss criterion, short-cycle-first heuristic, and repair) the BreakCycle algorithm. In practice, its complexity is much less than $O(n^2)$.

## 5. Network Structure Perturbations

To assess the quality of the obtained network, we perform local structural perturbations to assess the local stability of the obtained network $G$. Suppose we perturb $G$ into a new structure $\hat{G}$. Let the log-odds-ratio be

$$\Delta\ell = \ell(\hat{G}) - \ell(G) = \log\frac{P(D\,|\,\hat{G})}{P(D\,|\,G)}. \qquad (7)$$

We say $G$ is locally stable if $\Delta\ell < 0$. We consider the following two types of local perturbations.

### 5.1 Edge Perturbation

Here we attempt to perturb the edge $e_{ij} = g_i \rightarrow g_j$ to see if the Bayesian likelihood score is improved. If the score is improved, than the edge is "unstable". A "brute force" perturbation is to simply cut $e_{ij}$ and compute the log-odds-ratio of Eq.(7). However, after $e_{ij}$ is cut, $g_j$'s parent-set is no longer optimal. Thus this brute force perturbation will usually render an edge stable. For this reason, we do a soft perturbation. We use the K2+ algorithm to find the new optimal parents for $g_2$, excluding the cut edge (but keeping all other parents if any). We calculate $\Delta\ell_e^{EP}$ ("EP" stands for Edge Perturbation) and the percentage of stable edges

$$r^{EP} = \frac{1}{|E|}\sum_{e\in E}\delta(\Delta\ell_e^{EP}), \qquad (8)$$

where $\delta(x) = 1$ if $x \le 0$ and 0 otherwise. $r^{EP}$ is an indicator of the local stability of $G$. A stable $G$ should have $r^{EP} \sim 1$.

Note that EP is a local stability test because it produces a new locally optimal structure for comparison. The more negative $\Delta\ell_e^{EP}$, the more "stable" the edge $e$ is.

### 5.2 Improving Edge Stability

By perturbation, we can identify those unstable edges whose replacements lead to better likelihood scores. This suggests that we may improve the likelihood of the whole network by substituting these unstable edges with their replacements, subject to the constraint that no cycles appear after replacements.

The edge-stability-improvement algorithm first sorts the $\Delta\ell_e^{EP}$ of all unstable edges. Similar to the repair algorithm in §4.3, it then goes through all unstable edges following the sorted ordering (starting with the most unstable edge). For a given unstable edge $e$, the optimal replacement found in EP is first tried to see if there is cycle caused; if no, then the optimal replacement is used; otherwise the K2+ search algorithm is invoked to search the best replacement (similar to §4.3, the search is subject to the acyclic condition, and all other parents of the current node are retained.).

By applying the edge-stability-improvement algorithm, the Bayesian likelihood score of $G$ is improved while the number of unstable edges is reduced.

Note that our goal is to detect and repair unstable edges to improve a single structure. This differs from other edge quality assessments, e.g. averaging over a large number of structures [19], where the edge importance is not associated with a particular structure.

## 6. Summary of Our Approach

As a brief summary, we outline the major paradigm of our learning algorithm of Bayesian networks as follows.

(1) Use K2+ algorithm to generate the candidate graph.

(2) Use BreakCycle algorithm to generate DAG from the candidate graph, and repair the local structures to make the DAG locally optimal.

(3) Use structural perturbation EP to assess the local stability of the learnt BN, and improve the stability with the algorithm in §5.2.

## 7. Experiments

### 7.1 Data

We use two data sets in this paper. The first is the well-known Alarm data set [5], which consists of 37 variables and 10000 samples. There is an intrinsic ordering of these variables, which however is not used in our experiments since our major concern is how to model the data without the ordering information. We use the Alarm data accompanying the PowerConstructor package [2]. We compare our results with WinMine [4,15] developed by Microsoft, because it can generate DAGs without ordering of variables.

The second data set is the Rosetta Inpharmatics Compendium [14], which is a genome data set of yeast. The data have 481 real-valued gene variables with 300 data points (experiment conditions). The original real-valued variables are discretized to 3-states via thresholding at $\mu \pm 0.4\sigma$ ($\sigma$ -- standard deviation, $\mu$ -- mean). These states correspond to the over-expression, baseline, and under-expression of genes.

We use Cross Validation (CV) to evaluate the generalization strength of learnt BNs. We use 10-fold CV.

One metric for CV is the cross-validated likelihood of the testing data computed from the learnt BN where the parameters (i.e. conditional probability tables) are estimated from the training data. The following normalized logarithm likelihood $L_{CV}$ is used:

$$L_{CV} = \frac{1}{nN} \sum_{s=1}^{10} \sum_{g \in D_s} \log p(g \mid G), \qquad (9)$$

where $D_s$ is the $s$th-fold test set. Clearly, the larger $L_{CV}$, the better the BN characterizes the data. Hence, $L_{CV}$ evaluates how well the learnt BNs generalize to unseen data. The normalized data likelihood $L$ for learning is similarly computed, which is also used in [15].

### 7.2 Results on Alarm Data Set

The results on the Alarm data set are shown in Table 1. Results of our algorithms, WinMine, and ordering-space-search, are shown, together with those computed from the known true Alarm structure and the null model (i.e. without interconnecting edges). The $k$-max search in K2+ clearly improve the quality of the learnt BNs as seen from the improvements of quality-measures $\ell$, $L$, $L_{\mathrm{CV}}$ and $r^{EP}$ of the 3-max search results as compared with the 1-max (K2) search results. The edge stability algorithm of §5.2 clearly improves all the $\ell$, $L$, $L_{\mathrm{CV}}$, and $r^{EP}$. $r^{EP}$ becomes 1 afterwards.

Compared to true model results, our best results (i.e. 3-max with improved stability) are very close. Remember that the ordering of variables is assumed unknown, thus it is highly unlikely the true structure can be recovered from data. Hence, these results indicate our network can model the data almost equivalent to the true model, with a different network structure (57 edges in our model versus 46 edges in the true model).

We run WinMine using three different $\kappa$ values, 0.01 (default value), 0.002, and 8e-12, to adjust the network to have the same number of edges as our results or as the true model. For the first two $\kappa$ values, we obtained BNs with 57 edges, among which 55 are the same; the quality metrics of both structures are the same, but are not as good as our results. In the last case the BN has 46 edges, the same as the true model, however it has much worse performances than our models.

If an ordering of variables is known, we can run K2 to efficiently compute the structure. When no ordering information is known, one may generate a random ordering and compute the structure. One can generate many random orderings to search for best structure [7]. We perform this ordering-space-search experiment for 100 random trials. Both the mean and best results are listed in Table 1. They are substantially worse than both our and WinMine's results. This indicates that it is hard to generate network with good quality from random orderings of variables, even at great computation expense.

We compare the BN structure obtained using our algorithm with that obtained using WinMine, and also the true model (although due to the theory of equivalent graphs [11], this direct comparison is not necessarily the best comparison). For example, our BN and that of WinMine both have 57 directed edges, among which 39 are the same. The number of overlapping edges of our result and the true model is 31, while that of WinMine and the true model is 22.

*Table 1*. Results on the Alarm data set. ($\ell$, $L$, $L_{CV}$ are all normalized by $nN$; $\kappa$ is the WinMine parameter controlling the complexity of network structure.)

| Method | Parent Search Method | Learning (all data) | | | | CV (10-fold) |
|---|---|---|---|---|---|---|
| | | $\ell$ | $L$ | $r^{EP}$ | $\|E\|$ | $L_{CV}$ |
| Our method (Before improving stability) | 1-max | -0.2587 | -0.2543 | 0.9123 | 57 | -0.2554 |
| | 3-max | -0.2581 | -0.2539 | 0.9298 | 56 | -0.2550 |
| Our method (After improving stability) | 1-max | -0.2566 | -0.2522 | 1.0000 | 56 | -0.2533 |
| | 3-max | -0.2562 | -0.2519 | 1.0000 | 57 | -0.2530 |
| True Model | | -0.2555 | -0.2517 | 0.9783 | 46 | -0.2526 |
| WinMine | $\kappa$ = 0.01 (Default) | -0.2593 | -0.2551 | 1.0000 | 57 | -0.2561 |
| | $\kappa$ = 0.002 | -0.2593 | -0.2551 | 1.0000 | 57 | -0.2561 |
| | $\kappa$ = 8e-12 | -0.2655 | -0.2622 | 0.9783 | 46 | -0.2630 |
| Search of Ordering Space (100 trials) | Best results | -0.2633 | -0.2578 | 0.8730 | 63 | -0.2592 |
| | Mean results | -0.2701 ±0.0026 | -0.2631 ±0.0023 | 0.8765 ±0.0521 | 74.0 ±5.4 | -0.2650 ±0.0023 |
| Null Model | | -0.5822 | -0.5813 | --- | 0 | -0.5815 |

*Table 2*. Results on the Yeast gene expression data. $n$ is the number of iterations in stability enhancement.

| Search Method and Parameters | | | Learning (All Data) | | | | | CV (10-fold) |
|---|---|---|---|---|---|---|---|---|
| Our Method | $k$-max | $n$ | $\ell$ | $L$ | $r^{EP}$ | $\|E\|$ | $T$ (min) | $L_{CV}$ |
| | 1-max | 0 | -0.9770 | -0.8269 | 0.6222 | 1326 | 9 | -0.9420 |
| | | 1 | -0.9691 | -0.7831 | 0.8451 | 1517 | +44 | -0.9303 |
| | | 2 | -0.9662 | -0.7687 | 0.9426 | 1586 | +25 | -0.9255 |
| | | 3 | -0.9654 | -0.7640 | 0.9863 | 1611 | + 8.7 | -0.9242 |
| | | 4 | -0.9651 | -0.7627 | 0.9951 | 1620 | + 1.8 | -0.9237 |
| | 3-max | 0 | -0.9710 | -0.8019 | 0.7077 | 1433 | 16 | -0.9338 |
| | | 1 | -0.9654 | -0.7689 | 0.8901 | 1583 | +40 | -0.9250 |
| | | 2 | -0.9639 | -0.7624 | 0.9579 | 1614 | +27 | -0.9226 |
| | | 3 | -0.9635 | -0.7612 | 0.9821 | 1620 | +14 | -0.9220 |
| | | 4 | -0.9634 | -0.7606 | 0.9889 | 1624 | + 6.2 | -0.9218 |
| | | 5 | -0.9631 | -0.7588 | 0.9957 | 1634 | + 0.09 | -0.9212 |
| WinMine | $\kappa$ | | $\ell$ | $L$ | $r^{EP}$ | $\|E\|$ | $T$ (min) | $L_{CV}$ |
| | 0.01 (Default) | | -1.0218 | -0.9918 | 0.2868 | 272 | 57 | -1.0063 |
| | 0.50 | | -0.9916 | -0.9373 | 0.4944 | 627 | 115 | -0.9683 |
| | 0.99 | | -0.9644 | -0.7744 | 0.9064 | 1528 | 229 | -0.9239 |
| | 0.999 | | -0.9638 | -0.7591 | 0.9468 | 1616 | 235 | -0.9224 |
| | 1.00 | | -0.9636 | -0.7554 | 0.9494 | 1641 | 268 | -0.9220 |
| Null Model | | | -1.1079 | -1.0918 | --- | 0 | 0 | -1.0987 |

Overall, the true model has the best quality measures. This is of course expected. Our resultant networks have close performance to the true model, which are clearly better than WinMine and the ordering-space-search method. The null model performs considerably worse in all measures.

### 7.3 Results on Yeast Gene Expression Data Set

Table 2 compares the results on the yeast gene data. In both our BNs and WinMine's results, there are more than 1600 edges for the 481 nodes.

For learning, Table 2 shows that $k$-max search in K2+ improves $\ell$, $L$, $r^{EP}$ of learnt BNs. Applying the edge-stability-improvement algorithm leads to steady improvements in all quality-measures.

We also run WinMine for a variety of parameter $\kappa$. The best results are obtained by setting $\kappa$ to its maximal value, i.e. 1.0. Table 2 shows that in the best case, WinMine results are worse than that of 3-max, i. e. smaller training score $\ell$, smaller edge stability rate $r^{EP}$, and less generalization strength $L_{CV}$. It is interesting to see that the training likelihood $L$ of WinMine result is higher than that of 3-max, however $L_{CV}$ of WinMine is lower than that of 3-max; this implies that the best net-

*Table 4*. Examples of stable associations in our results; their biological plausibility are indicated by the summaries in the SGD at Stanford University.

| Associations | Biological significance |
|---|---|
| ADE1 → ADE2 | ADE1 is a phosphoribosylaminoimidazole-succinocarboxamide synthase while ADE2 is a phosphoribosylaminoimidazole carboxylase. |
| ARG1 → ARG5 | Both genes are related to arginine biosynthesis. ARG1 is an argininosuccinate synthase while ARG5 is an ornithine carbamoyltransferase. |
| ERG3 → ERG25 | ERG3 is a C-5 sterol desaturase; ERG25 is a C-4 methyl sterol oxidase; both are related to ergosterol biosynthesis. |
| HXK1 → GPH1 | HXK1 is a hexokinase related to fructose metabolism and glycolysis while GPH1 is a glycine amidinotransferase. |
| HXT6 → HXT7 | Both genes are fructose transporter, glucose transporter, and mannose transporter that are related to hexose transport. |
| IDH2 → IDH1 | Both genes are related to glutamate biosynthesis, isocitrate metabolism, and tricarboxylic acid cycle. Both are isocitrate dehydrogenases (NAD+). |
| IDH2 → ACO1 | IDH2 is a isocitrate dehydrogenase and ACO1 is an aconitate hydrataseand. Both are related to glutamate biosynthesis. |
| MET16 → MET10 | Both genes are related to methionine metabolism and sulfate assimilation. MET10 is a sulfite reductase (NADPH) while MET16 is a phosphoadenylyl-sulfate reductase (thioredoxin). |
| RNR2 → RNR4 | Both are ribonucleoside-diphosphate reductases that are related to DNA replication. |
| SIT1 → FET3 | FET3 is a multicopper ferroxidase iron transport mediator; SIT1 is a siderochrome-iron (ferrioxamine) uptake transporter. Both are related to iron transport. |
| YPS6 → YPS5 | Both are aspartic-type endopeptidases that are related to cell wall (sensu Fungi). |

work of WinMine might overfit data slightly.

Table 2 also lists the time (on PIII 1G CPU) of each method. A plus "+" in our results means the time spent for the current step for edge stability improvement. (Our algorithms were implemented in Matlab and C++, while WinMine was implemented in C++). Our method uses less than 16 minutes to generate an initial BN, and 1 hour or so to refine the network structure. In contrast, WinMine takes about 4 hours to generate a network with the similar performance. These timing results show that our methods are much faster than WinMine, due to our algorithm's $O(n^2)$ computational complexity.

Many of regulatory relationships with high stability obtained in the networks can be confirmed by the biological literatures. We extract sub-networks involving well-studied genes. We find these genes often have closely related biological functions. These biologically plausible regulatory relationships and sub-networks indicate the strength of our new methods. Table 4 briefly lists several highly stable regulatory relationships we have obtained, which are also supported by biological findings. An arrow between two genes, $g_i \rightarrow g_j$, implies $g_i$ likely regulates $g_j$. From public-domain yeast-gene databases, e.g. the *Saccharomyces* Genome Database (SGD) at Stanford University [13], the biological significance of the associations can be easily confirmed. For example, in our results the gene ARG1 regulates the gene ARG5. Biologically, they are both related to arginine biosynthesis [13].

## 8. Discussions and Conclusion

A characteristic of the networks in our results is that they are rather sparse, which partially explains the high local stability of the obtained structures regarding to the perturbations. This also implies that some weaker or higher order associations could be missed in our method. One way to capture those associations is to let K2+ parent-search algorithm generate denser candidate graph, by relaxing the conditional independence requirement and/or expanding the search to generate large parent sets.

Currently, our cycle elimination algorithm deterministically returns a DAG given a candidate graph based on the least likelihood loss criterion. One could also introduce randomization so that the algorithm returns a number of structures that can facilitate model averaging.

In this paper, we use local structural perturbations to systematically assess the roles of individual edges in the network. Based on them, one could build larger subnet-level perturbations using clustering, seed growing, etc. This could help to detect sub-structures of Bayesian networks.

We use structural perturbation to define edge importance and cross validation for quality assessment. Note that there is possibility of overfitting the underlying data distribution. The cross-validation method provides means to see if the model overfits the data. Our structural perturbations provide potentially useful assessment of this issue. As shown in the experiments, stable networks against structural perturbations are not likely to be overfitted.

Another method is the Bayesian model averaging [8,19] that produces confidence scores for individual edges by averaging over high scoring structures. When the number of observations is small compared to the number of variables, there is concern that the data is insufficient to distinguish among high scoring structures with confidence. This question could be probed by techniques like our methods, model averaging, etc.

In summary, we propose improved techniques for efficiently learning BN for large number of variables.

We expand the Bayesian network search space while reducing search for less likely space due to conditional independence. This constructs locally optimal parent-set for each node to form the candidate graph. A systematic approach based on minimal-likelihood-loss is used to eliminate the cycles in the candidate graphs, which is efficiently implemented using a short-cycle-first heuristic. The maximal-loss-first heuristic is used to repair the local structures to assure the local optimality of the DAG. Edge perturbation is proposed to evaluate the local stability of the network. Unstable edges are replaced, leading to more stable network. Cross-validation is used to assess the generalization strength of the network. Based on experiments on Alarm data and a genomic data set, the quality of the networks learnt is better than those of WinMine and ordering-space-search, while the speed is improved significantly. The associations learnt from the genomic data are in good agreement with the earlier findings of biological literature.

## Acknowledgements

We thank Edward Herskovits for discussions on Bayesian learning, David M. Chickering for discussion on using WinMine, and Dana Pe'er for providing the list of 481 genes. The software and network structure results are available upon request. This work is supported by Department of Energy, Office of Science (MICS Office and a LBNL LDRD) under contract No. DE-AC03-76SF00098.

## References

1. Buntine, W, A guide to the literature on learning probabilistic networks from data, IEEE Trans on KDE, vol.8, no.2, pp.195-210, 1996.
2. Cheng, J., Bell, DA, Liu, W., Learning belief networks from data: an information theory based approach, 6th ACM Int'l Conf on Information and Knowledge Management, 1997.
3. Chickering, D., Geiger, D., and Heckerman, D., Learning Bayesian Networks is NP-Hard, Technical Report MSR-TR-94-17, Microsoft Research, 1994.
4. Chickering, D.M., The WinMine toolkit, MSR-TR-2002-103, Microsoft Research, Oct. 2002.
5. Cooper, G.F., and Herskovits, E., A Bayesian method for the induction of probabilistic networks from data, Machine Learning, vol.9, pp. 309-347, 1992.
6. Corman, T.H., Leiserson, C.E., and Rivest, R.L., Introduction to Algorithms, MIT, 2001.
7. Friedman, N., and Koller, D., Being Bayesian about network structure: a Bayesian approach to structure discovery in Bayesian networks, Machine Learning, 2002.
8. Friedman, N., Linial, M., Nachman, I., and Pe'er., D., Using Bayesian networks to analyze expression data, J. of Comp. Biology, vol.7, pp.601-620, 2000.
9. Harary, F., Graph Theory, Addison-Wesley, Reading, Mass, 1969.
10. Hartemink, A.J., Gifford, D.K., Jaakkola, T.S., and Young, R.A., Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks, PSB-2001, Jan 2001.
11. Heckerman, D. A tutorial on learning with Bayesian networks, in M.I. Jordan (Ed.) *Learning in Graphical Models*: 301-354, MIT Press, 2000.
12. Heckerman, D., Chickering, D.M., Meek, C., Rounthwaite, R., and Kadie, C., Dependency networks for inference, collaborative filtering, and data visualization, J. Machine Learning Research, vol.1, pp.49-75, 2000.
13. http://genome-www.stanford.edu/Saccharomyces.
14. Hughes, T.R., et al, Functional discovery via a compendium of expression profiles, Cell, vol.102, pp.109-126, 2000.
15. Hulten, G., Chickering, D.M., and Heckerman, D., Learning Bayesian networks from dependency networks: a preliminary study, AI & Statistics 2003, pp.54-61, Key West, Florida, Jan. 2003.
16. Larranage, P., Poza, M., Yurramendi, Y., Murga, R.H., and Kuijpers, C.M., Structural learning of Bayesian networks by genetic algorithms: a performance analysis of control parameters, IEEE Trans. PAMI, vol.18, pp.912-926, 1996.
17. Murphy, K., and Mian, S., Modeling gene expression data using dynamic Bayesian networks, Technical report, CS Division, Berkeley, 1999.
18. Pearl, J., Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, San Mateo, CA: Morgan Kaufmann, 1988.
19. Pe'er, D., Regev, A., Elidan, G., and Friedman, N., Inferring subnetworks from perturbed expression profiles, Bioinformatics, vol. 17, pp.215S-224S, 2001.
20. Peng, H.C., Herskovits E, and Davatzikos C, Bayesian clustering methods for morphological analysis of MR images, IEEE Int'l Symp on Medical Imaging: Macro to Nano, pp.875-878, Washington. D.C., July, 2002.
21. Robinson, R.W., Counting labeled acyclic graphs, In C.H.C. Little (Ed.), Lecture Notes in Mathematics, 622: Combinatorial Mathematics V., New York: Springer-Verlag, 1977.
22. Sarkar, S. and Boyer, K. L., "Integration, Inference, and Management of Spatial Information Using Bayesian Networks: Perceptual Organization" IEEE Trans on PAMI, vol.15, no.3, pp.256-274, 1993.
23. Yoo, C., Thorsson, V., and Cooper, G., Discovery of causal relationships in gene regulation pathways from a mixture of experimental and observational DNA microarray data, PSB-2002, pp. 498-509, 2002.